Sensors and Materials, Vol. 38, No. 4 (2026) 1859–1877
MYU Tokyo

S & M 4411

Development of an Intelligent Wheelchair System Combining D* Lite and Dynamic Window Approach Path Planning Algorithms

Neng-Sheng Pai, Chun-Chieh Chang, Pi-Yun Chen,* and Xi-Zhi Huang

Department of Electrical Engineering, National Chin-Yi University of Technology,
Taichung City 41170, Taiwan (ROC)

(Received May 28, 2025; accepted January 13, 2026)

Keywords: autonomous navigation, intelligent wheelchair, Simultaneous Localization and Mapping (SLAM), D* Lite, Dynamic Window Approach (DWA), path planning

In this paper, we aim to develop an intelligent wheelchair system with autonomous navigation capabilities, integrating functions such as localization, environment mapping, autonomous navigation, and obstacle avoidance to reduce the burden on healthcare personnel in assisting individuals with mobility issues. First, a conventional electric wheelchair is modified to receive external speed commands, transforming it into an intelligent wheelchair equipped with various sensors to perceive environmental information, laying the foundation for subsequent navigation technologies. Then, the system uses Simultaneous Localization and Mapping technology, combined with the Real-time Appearance-based Mapping algorithm, to integrate sensor data and construct accurate environmental maps. We found that using either a depth camera or Light Detection and Ranging alone has limitations, so this system combines both to improve the accuracy and stability of mapping. Finally, on the basis of the constructed environment map, the D* Lite algorithm is applied for global path planning to determine the optimal route, while the Dynamic Window Approach algorithm is used for local path planning, dynamically adjusting the intelligent wheelchair's route to address real-time obstacle avoidance. Experimental results confirm that the system can effectively assist individuals with mobility impairments in reaching their destinations autonomously, demonstrating stable and safe navigation performance.

1. Introduction

With advances in technology and improvements in healthcare standards, the average human lifespan has continued to increase—from 64.6 years in the early 1990s to 72.8 years in 2019, and it is projected to reach 77.2 years by 2050.[1] However, this achievement also brings about the issue of population aging, which adds pressure to national economies, social welfare systems, and healthcare services. According to the *2022 Revision of World Population Prospects* by the United Nations Department of Economic and Social Affairs, the proportion of the global population aged 65 and over is expected to rise from 10% in 2022 to 16% by 2050. At that point,

*Corresponding author: e-mail: chenby@ncut.edu.tw
https://doi.org/10.18494/SAM5757

ISSN 0914-4935
https://myukk.org/

the elderly population will be twice as large as the number of children under the age of five, and by 2030, one in six people worldwide is expected to be over the age of 60.[2] In light of the growing shortage of long-term care personnel, there is an urgent need to adopt technological solutions in healthcare and caregiving.

The intelligent wheelchair can be regarded as a type of Autonomous Mobile Robot (AMR), offering considerable potential to enhance the quality of life for individuals with mobility impairments and to improve the operational efficiency of long-term care facilities. Typically equipped with Light Detection and Ranging (LiDAR), cameras, and an Inertial Measurement Unit (IMU), such systems utilize Simultaneous Localization and Mapping (SLAM) techniques to construct environmental maps and estimate the robot's position using sensor data. Path planning algorithms are then employed to determine the optimal route, enabling the intelligent wheelchair to autonomously navigate to designated locations and achieve a complete process from environmental perception to autonomous movement. This type of system can significantly improve the quality of life for elderly and mobility-impaired individuals while effectively enhancing the utilization of human resources in long-term care institutions.

2. Related Work

This section is divided into three parts for review and discussion: sensor, SLAM, and path planning algorithms.

2.1 Sensor

AMRs rely on various sensors to perceive environmental information and changes, enabling them to operate flexibly in complex environments. In 2022, Tesla announced that it would rely entirely on vision sensors to achieve autonomous driving. However, to provide a 360° field of view, multiple cameras are required to eliminate blind spots.[3] Vision sensors, however, are highly susceptible to lighting interference, making them unsuitable for stable operation in outdoor environments. According to research by Milford and George, robots using only vision sensors may suffer from image blur and unstable exposure when driving at high speeds over uneven terrain due to vibrations.[4] These issues underscore the challenges faced by systems that rely exclusively on vision sensors.

2.2 SLAM

The concept of SLAM was first proposed by Smith and Cheeseman, using LiDAR as the main sensor.[5] However, since LiDAR is expensive and cannot obtain semantic information, subsequent research began to turn to other sensors. Among them, Longuet-Higgins proposed an algorithm suitable for depth cameras, which promoted the development of SLAM technology.[6] Currently, SLAM is divided into Gmapping[7] and Hector SLAM,[8] which mainly use LiDAR, and ORB-SLAM[9] and Real-time Appearance-based Mapping (RTAB-Map),[10] which mainly use cameras. Gmapping calculates the map by sampling a large number of particles and fitting

them to the target distribution. Repeated sampling and calculation consume many computing resources, so Gmapping is more suitable for small indoor spaces. Hector SLAM aligns LiDAR data with maps through the Gauss–Newton method. This algorithm is suitable for uneven terrain, but has high requirements on the update frequency and measurement noise of the LiDAR. ORB-SLAM uses visual odometry to construct maps and the Bundle Adjustment[11] method for optimization to achieve a higher accuracy than other methods. However, this method is prone to accumulating errors when constructing unknown maps, and when rotating at high speed, the image obtained by the camera will become blurred, thus affecting the quality of the constructed map. However, RTAB-Map introduces a three-level storage management mechanism that controls the data size of each calculation call, saving computing resources compared with other methods. It can also accept data input from cameras and LiDARs, combining the advantages of the two sensors and making up for their shortcomings. On the basis of the above characteristics, in this paper, we chose RTAB-Map as the mapping solution.

2.3 Path planning algorithms

Path planning algorithms are divided into global and local path planning according to different applicable environments. Global path planning plans the path on the basis of the overall environmental information on the map. Representative algorithms include the A*[12] and D* Lite[13] algorithms. The A* algorithm evaluates the suitability of a node by calculating the sum of the distance from the starting point to the current position and the distance from the current position to the goal. The D* Lite algorithm is based on the improved Lifelong Planning A*[14] algorithm, which searches for nodes in reverse from the goal to the starting point. Even if unexpected obstacles appear on the path, the path can be quickly updated using previously calculated node distance information. Local path planning uses data from current sensors for planning, where the Dynamic Window Approach (DWA)[15] algorithm selects the local optimal path through motion constraints, taking into account chassis performance and providing a local path with low computational complexity and fast response. However, if we only rely on the local path planning algorithm without combining the global path planning algorithm, we will not be able to fully consider the overall environmental information. To obtain the best path planning, we selected D* Lite as the global path planning algorithm and combined it with the DWA local path planning algorithm to achieve good stability and obstacle avoidance capabilities.

3. Methodology

The system architecture of this paper is mainly divided into three parts, as shown in Fig. 1. The first part is the design and construction of an intelligent wheelchair, which uses a traditional electric wheelchair as the basic platform and integrates an edge computing platform as the core control unit, equipped with sensors such as binocular depth cameras, IMU, and 2D LiDAR. Through the application of these sensing components, traditional electric wheelchairs can be upgraded to mobile platforms with intelligent functions. They can receive external control commands and have environmental perception capabilities to identify surrounding objects and

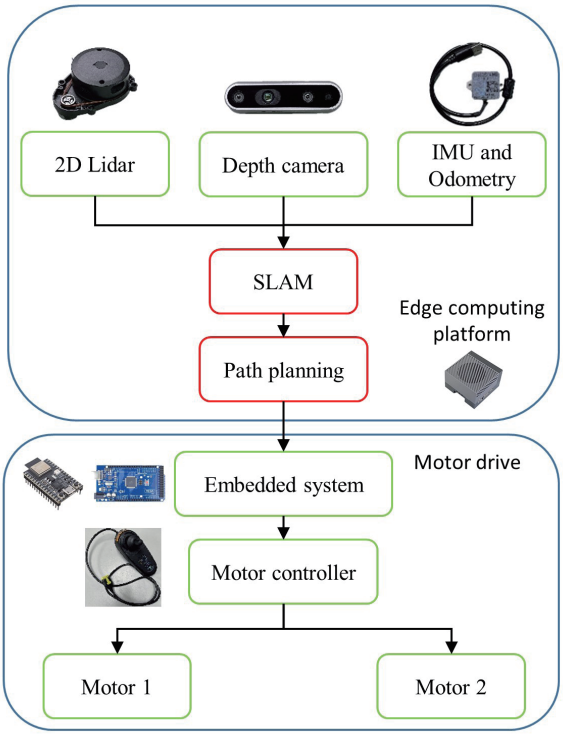


Fig. 1. (Color online) System architecture diagram.

obstacles. The second part is map construction and path planning. Through the sensor data collected by the LiDAR, depth camera, and odometer, the environmental map is generated through the SLAM algorithm, and the target points are specified for the path planning algorithm to use, so as to plan the best travel path from the intelligent wheelchair location to the destination. The third part is the motion control module. The edge computing platform sends speed commands to the embedded platform according to the optimal path. The embedded platform then converts the obtained speed commands into motor control signals to drive the motors on the left and right sides (motors 1 and 2) to achieve the autonomous movement and navigation of the intelligent wheelchair.

3.1 Construction and design of intelligent wheelchair

In this section, we explain how to upgrade a traditional electric wheelchair to an intelligent wheelchair with autonomous navigation capabilities. It is mainly divided into two parts: sensor integration and chassis control system modification. In terms of sensors, since intelligent wheelchairs need to travel outdoors, depth cameras alone are not effective in dealing with harsh environments, and LiDARs need to be installed to increase the robustness of mapping. The working principle of LiDAR is to drive the laser and receiver through a rotating mechanism, so that the beam emitted by the laser returns to the receiver through reflective obstacles, and then the signal processing unit calculates the distance to surrounding objects, so it is not affected by

light. In addition, the system also integrates an IMU to record the movement direction and posture information of the intelligent wheelchair. To improve the accuracy and stability of system positioning, a Kalman filter[16] is used to process noise during the measurement process and estimate the accurate state of the system on the basis of the changes between the current and previous states of the system, thereby reducing errors and improving the stability of the system. In terms of chassis control, the original electric wheelchair uses joystick control, which controls the rotation speeds of the left and right rear wheels by changing the Hall voltage generated by the position of the joystick. To realize the intelligent control function, the original control circuit must be modified. First, remove the joystick module and use the embedded platform to output the corresponding DC voltage signal to simulate joystick control. Experimental observations revealed that the joystick is connected to the main circuit board through eight pins for power supply and communication, as shown in Fig. 2.

The motor control signal is implemented through the embedded platform's Dac_{write} function, which enables the embedded platform to output a DC voltage from 0 to 3.3 volts by setting the lowest voltage to 0 volts and the highest voltage to 3.3 volts due to the DC voltage output limitation of the embedded platform. When the motor is at rest, the speed is defined as 0%, and the maximum linear speed at 3.3 volts is defined as 100%. Within this range, by collecting the change data of the motor speed under different input voltages and fitting the data using polynomial curve fitting, the relationship between the motor speed and the input voltage can be obtained. The relationship is shown in Eqs. (1) and (2), where Dac_{left} and Dac_{right} represent the values required to be input into the Dac_{write} function, and V_{left} and V_{right} are the desired speeds of the left and right wheels, respectively, in meters per second.

$$Dac_{left} = 28.2122 \times V_{left} + 30.1732 \times V_{right} \tag{1}$$

$$Dac_{right} = 57.0194 \times V_{left} - 58.6717 \times V_{right} \tag{2}$$

3.2 SLAM based on real-time appearance-based mapping

RTAB-Map is a SLAM algorithm widely used in mapping and navigation. It has a complete system architecture and high stability, and supports loop closure detection based on visual

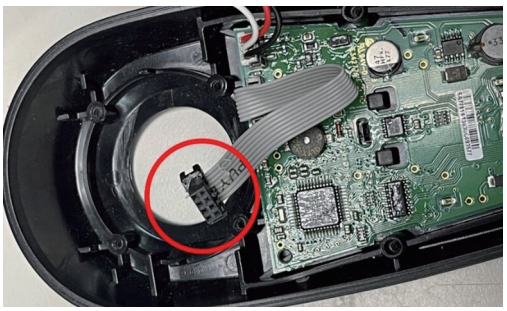


Fig. 2. (Color online) Diagram of the controller circuit board.

features. The algorithm organizes the map in the form of a graph structure with multiple nodes and connections. A node is created for each frame of visual sensor data. The node content includes information such as the values of all sensors, odometer posture, a local map, and visual words used for detection. The architecture of RTAB-Map is shown in Fig. 3.[17] After the sensor publishes data, it will go through the steps of synchronization, Short-Term Memory (STM), loop closure detection, graph optimization, and global map assembly, and finally, it will generate a grid map that can be used for navigation.

3.2.1 Synchronization

When receiving data from various sensors, since each sensor may not publish data at the same speed and time, the first step in inputting sensor data into RTAB-Map is to synchronize the data to avoid errors caused by inconsistent data. There are two types of synchronization: exact and approximate. An exact synchronization requires that the input data have exactly the same timestamp and is applicable to data from the same sensor, such as images from the left and right lenses of a binocular camera. An approximate synchronization compares the timestamps of the input data and attempts to synchronize all the data with minimal latency error. This approach is suitable for data between different sensors, such as the data output by cameras and LiDARs.

3.2.2 Storage management mechanism

To improve computing efficiency, RTAB-Map adopts a three-level storage management mechanism. Each time a closed loop detection calculation is performed, only the nodes located in STM and Working Memory (WM) will be calculated. Initially, the newly acquired nodes will be placed in STM. The storage limit of STM can be adjusted according to the performance of the edge computing platform, the walking speed, and the frequency of positioning point acquisition. Taking the above factors into consideration, in this paper, we set the upper limit to 10 nodes. When the number of nodes in STM reaches the upper limit, the oldest node will be moved to WM. If the processing time for loop closure detection is very long, the nodes in WM will be

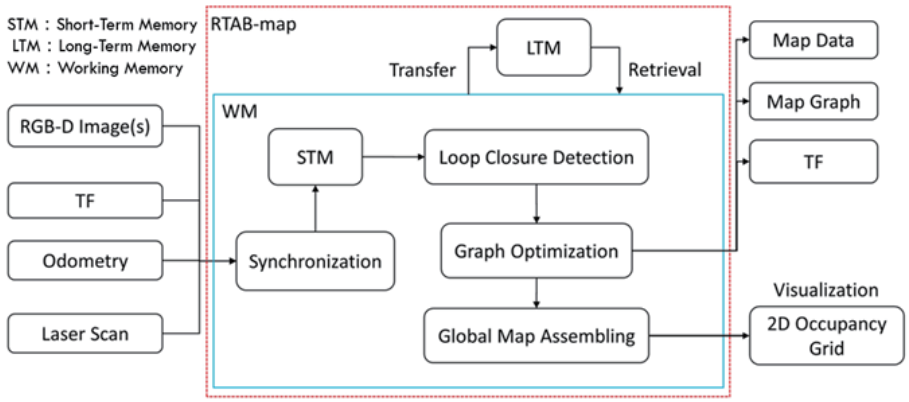


Fig. 3. (Color online) RTAB-Map architecture diagram.

transferred to Long-term Memory (LTM). The nodes in LTM will not participate in the calculation of loop closure detection, but when necessary, the nodes in LTM can be moved back to WM through the retrieval mechanism to participate in the calculation again.

3.2.3 Loop closure detection

When the intelligent wheelchair is constantly moving in the environment and updating the map, there may be incoherent or erroneous parts of the map due to repeated or similar scenes in the environment or due to odometer errors. Loop closure detection detects these repeated scenes by comparing the current and previous features. This is carried out by extracting visual vocabulary vectors from the image using the Speeded Up Robust Feature (SURF)[18] algorithm through the Bag-of-Words (BoW)[19] method to represent the feature points of the image. The steps of the BoW method are as follows:

1. Use the SURF algorithm to extract feature points in the image.
2. Build a dictionary and mark each feature as a word, then use the K-Means clustering algorithm[20] to merge words with similar meanings, and finally represent them in a word list.
3. Count the number of occurrences of feature points represented by each word in each image and obtain a feature histogram for comparison.

When these features are found to have high similarity, it means that it may have circled back to an area it has passed through before. Therefore, loop closure detection will identify whether loop closure occurs and make corrections to maintain the accuracy and quality of the map. In addition, RTAB-Map will dynamically evaluate the impact of nodes on the overall computing time. If the relevant nodes are not in WM, the relevant nodes will be retrieved from LTM to WM for subsequent calculations. However, if the processing time exceeds 1 s, the node with the lowest weight will be transferred to LTM. If there are multiple nodes with the same weight, the transfer will start from the node that joined first.

3.2.4 Graph optimization

Since errors may occur during loop closure detection or in the process of transferring or retrieving nodes, some very similar locations may cause the algorithm to produce false loop closure detections, which in turn introduces more errors into the map. Therefore, graph optimization methods are used to reduce errors in mapping. In this study, we used the Georgia Tech Smoothing and Mapping (GTSAM)[21] algorithm, which has the advantage of fast convergence compared with other algorithms.

3.2.5 Global map assembly

When new nodes are added to the map, the new local grid is combined with the existing global grid and the map is updated, including adding or removing obstacles. When a closed loop is detected, the global grid is reassembled according to the results of graph optimization, the nodes and connections in the map are rearranged, and finally, a complete global map is output to support subsequent navigation and planning tasks.

3.3 Path planning based on D* Lite and Dynamic Window Approach

After the map is constructed, path planning can be carried out. Its main goal is to calculate the best path from the starting point to the target point, which not only needs to meet navigation requirements, but also needs to avoid obstacles and cope with the challenges of dynamic environments. In this paper, we divided path planning into two levels: global and local. Global path planning calculates the best path to the target point on the basis of the global map. However, repeated global path planning consumes many computing resources, so the global path is only used as a reference for the approximate direction. The specific path and dynamic obstacle avoidance are planned by local path planning, which reduces the consumption of computing resources by calculating only a small section of the path each time. Since local path planning can update the path frequently, it has more advantages in dynamic environments.

The specific process of the path planning algorithm is shown in Fig. 4. First, after receiving the location of the goal, the global path planning algorithm is started to calculate the optimal path. Then, whether the goal has been reached is determined. If the destination has been reached, the path planning process ends; otherwise, the local path planning algorithm is started to replan the previously planned optimal path in sections. During this process, if there are no obstacles on the path, the intelligent wheelchair will start moving; if there are obstacles, the local path will be replanned until it reaches the goal.

3.3.1 D* Lite

The D* Lite algorithm is different from other path planning algorithms in that its path is calculated from the target point to the starting point, and the concept is to assume that the unknown areas in the environment are free space. By calculating the distance between a node and its predecessor and successor, the successor refers to all the nodes that can be reached by this node, and the predecessor refers to the previous node that can reach these nodes, as shown in

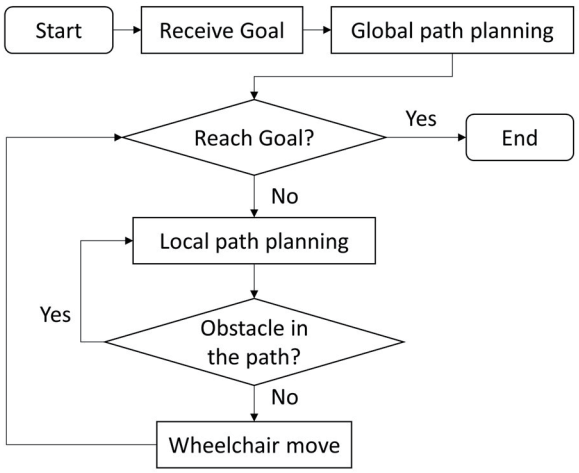


Fig. 4. Path planning flowchart.

Fig. 5. The green node can reach two red nodes, so the two red nodes are the successors of the green node and the green node is the predecessor of the two red nodes. Finally, the path with the smallest g value for all the nodes along the way is selected as the final path, where g is calculated as shown in Eq. (3), s' represents the next successor point of s, $c\left(s,s^{'}\right)$ is the distance from node s to s', and s_{goal} is the goal.

$$g(s)=\begin{cases}0, & \text{if } s=s_{goal}\\ \min_{s'\in suss(s)}\left(g(s')+c\left(s,s^{'}\right)\right), & \text{otherwise}\end{cases} \tag{3}$$

Because g represents the shortest distance from a node to the goal, if the node g value is updated frequently, it will affect the g value of adjacent nodes, resulting in a large waste of computing resources. Therefore, D* Lite introduces the right-hand side (*rhs*) value as the estimated distance from the current node to the goal and updates g on the basis of *rhs*. *rhs* is calculated as shown in Eq. (4), s' represents the next successor point of s, $c\left(s,s^{'}\right)$ is the distance from node s to s', and s_{goal} is the goal.

$$rhs(s)=\begin{cases}0, & \text{if } s=s_{goal}\\ \min_{s'\in pred(s)}\left(g(s')+c\left(s,s^{'}\right)\right), & \text{otherwise}\end{cases} \tag{4}$$

After calculating *rhs*, it is necessary to determine the local consistency state of the node to decide whether to update g. These states are divided into three types: Locally Consistent, Locally Overconsistent, and Locally Underconsistent, and they correspond to different processing methods; the flow chart is shown in Fig. 6.

Locally Consistent: When g is equal to *rhs*, it means that the current path is no different from the shortest path, and this stable state will be maintained without interference from external factors.

Locally Overconsistent: When g is greater than *rhs*, it means that the updated *rhs* is less than g, usually because obstacles on the path are cleared or a shorter path is found. At this point, the algorithm assigns the *rhs* value to the g value and updates the g values of the adjacent nodes and their subsequent nodes so that the algorithm can return to a locally consistent state.

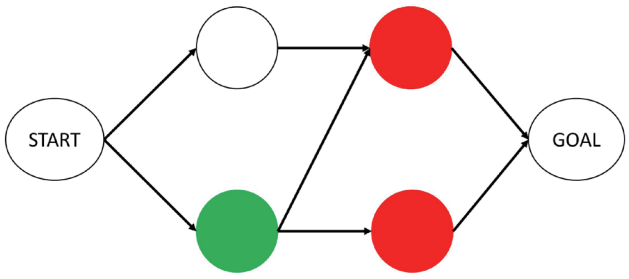


Fig. 5. (Color online) Predecessor and successor points diagram.

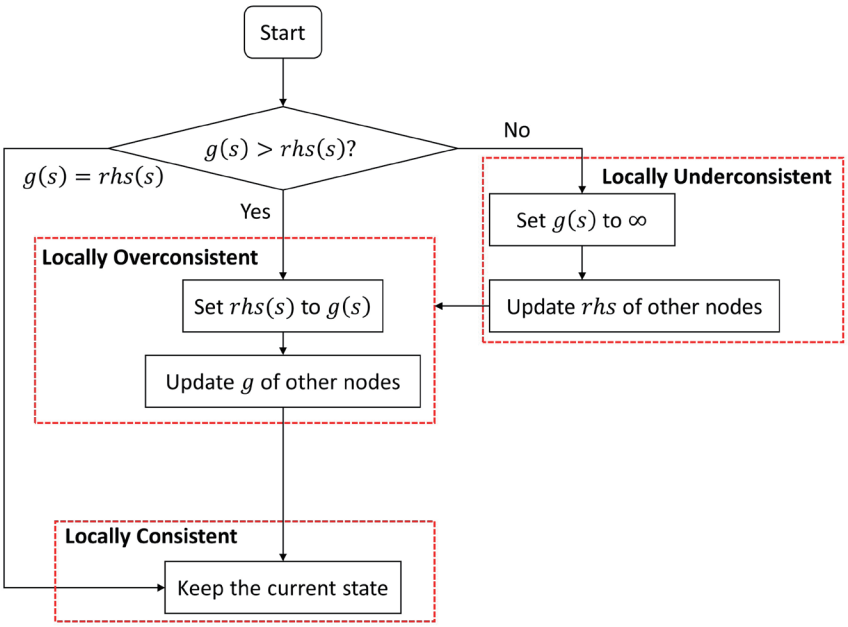


Fig. 6. (Color online) Locally consistent state assessment flowchart.

Locally Underconsistent: When *g* is less than *rhs*, it means that the originally planned path is affected by the sudden appearance of obstacles, making it no longer the shortest path or even impassable. At this time, the algorithm will set the *g* value of the node to infinity and recalculate the *rhs* values of adjacent and subsequent nodes, so that it enters a locally overconsistent state and obtains the latest path through the local overconsistent processing method, and finally restores to a stable locally consistent state.

According to Eq. (3), we can determine that $g(s')$ is needed to calculate the *rhs* value for the first time, and the *rhs* value at the goal is 0. Therefore, when performing calculations, the *rhs* values of all the nodes other than the goal and the *g* values of all the nodes including the goal are set to infinity. In this way, the goal is in a locally overconsistent state (i.e., the *g* value of the goal is greater than the *rhs* value). According to the processing method of this state, the *rhs* value of the goal is assigned to the *g* value, and it affects the nodes around it. This process is repeated a number of times, and finally, the *g* and *rhs* values corresponding to any node can be obtained.

However, the path update process still involves many node calculations that are not related to the shortest path. Therefore, two key values, k_1 and k_2, are introduced to determine the priority of the node. k_1 is computed as

$$k_1(s) = \min\left(g(s), rhs(s)\right) + h(s, s_{start}). \tag{5}$$

Here, $h(s, s_{start})$ is shown as

$$h\left(s, s_{start}\right)=\begin{cases}0, & \text{if } s=s_{start} \\ c\left(s', s\right)+h\left(s', s_{goal}\right), & \text{otherwise}.\end{cases} \tag{6}$$

The smaller the k_1 value of a node is, the closer the distance from the goal to the node and then to the starting point is to the straight-line distance, and the node can be determined to be a node with a shorter distance. However, if we simply compare the k_1 value, there may be multiple nodes with the same value. Therefore, we use k_2 to determine which node is closer to the goal. k_2 is defined as

$$k_2\left(s\right)=\min\left(g\left(s\right), rhs\left(s\right)\right). \tag{7}$$

The smaller the k_2 value is, the closer the node is to the goal. Therefore, when judging the priority of a node, the principle that the smaller the key value, the higher the priority is uniformly adopted.

3.3.2 Dynamic Window Approach

The principle of DWA is to sample multiple sets of target speeds (v, ω) in the speed space, simulate the motion trajectories of these speeds within a certain period of time, and then select the best trajectory through the evaluation function to drive the intelligent wheelchair. V_s is the set of all speeds that the intelligent wheelchair can reach, as shown in Eq. (8), V_s is affected by the maximum linear velocity v_{max}, the minimum linear velocity v_{min}, the maximum angular velocity ω_{max}, and the minimum angular velocity ω_{min}.

$$V_s=\left\{\left(v,\omega\right)\mid v\epsilon\left[v_{min}, v_{max}\right]\wedge\omega\epsilon\left[\omega_{min}, \omega_{max}\right]\right\} \tag{8}$$

Next, we define the acceleration space V_d. Under the current velocity v_c and the angular velocity ω_c, the velocity that can be achieved by the maximum acceleration $\dot{v}_a$, $\dot{\omega}_a$ and the maximum deceleration $\dot{v}_b$, $\dot{\omega}_b$ will be retained. V_d is defined as

$$V_d=\left\{\left(v,\omega\right)\mid v\epsilon\left[v_c-\dot{v}_b\Delta t, v_c+\dot{v}_a\Delta t\right]\wedge\omega\epsilon\left[\omega_c-\dot{\omega}_b\Delta t, \omega_c+\dot{\omega}_a\Delta t\right]\right\}. \tag{9}$$

For the intelligent wheelchair to stop successfully before hitting an obstacle, the target speed must satisfy condition (10). *dist*(v, ω) represents the shortest distance to the obstacle on the trajectory corresponding to the target speed (v, ω).

$$V_a=\left\{\left(v,\omega\right)\mid v\le\sqrt{2dist\left(v,\omega\right)\dot{v}_b}\wedge\omega\le\sqrt{2dist\left(v,\omega\right)\dot{\omega}_b}\right\} \tag{10}$$

Under the constraints of Eqs. (8)–(10), a range will be generated in the velocity space, which varies with the linear and angular accelerations of the motor. Then, an evaluation function is used to score these motion trajectories, and the motion trajectory with the highest score is selected to drive the intelligent wheelchair forward. The evaluation function is shown as

$$Score(v,\omega) = \eta\left(\alpha \times heading(v,\omega) + \beta \times dist(v,\omega) + \gamma \times vel(v,\omega)\right). \tag{11}$$

Here, $heading(v, \omega)$ is an azimuth evaluation function, which is used to evaluate the angle difference between the direction of the trajectory end of the intelligent wheelchair at the current speed and the target point. We do not want the intelligent wheelchair to make a turn with too large an angle, so the smaller the angle difference, the higher the trajectory evaluation. $dist(v, \omega)$ is responsible for evaluating the distance between the predicted trajectory and the nearest obstacle on the map. If the trajectory is too close to the obstacle, it may cause a collision, so the farther the trajectory, the higher the evaluation. $vel(v, \omega)$ represents the linear and angular velocites of the current intelligent wheelchair. Since we do not want the speed to be very slow, the higher the speed, the higher the evaluation. α, β, and γ are weights that can be adjusted arbitrarily according to task requirements. Here, these weights are all set to 1 to balance the effects of various evaluation factors. η is a normalization parameter. Since the calculation results of the three what cannot be added directly, the individual functions need to be normalized before they can be added.

4. Experiments

The final appearance of the intelligent wheelchair is shown in Fig. 7. It uses a depth camera, LiDAR, and IMU as the main sensors and RTAB-Map to construct a 2D grid map. Then, D* Lite and DWA path planning algorithms are used for autonomous navigation. The main experimental scenes are the second floor and basement of the Engineering Building.

4.1 Mapping results

In this section, we will show the 2D grid map created by RTAB-Map and integrate the map data and charts of data such as the average time for feature point extraction, graph optimization time, the time spent on creating each frame, the number of words contained in each frame, and node creation time.

Scene 1: The mapping result of the second floor of the Engineering Building is shown in Fig. 8, the actual appearance of the scene is shown in Fig. 9, and the statistical data related to the mapping are shown in Table 1. During the process, the intelligent wheelchair returned to the area where the map had been built three times, so it was judged that the loop was successfully closed three times.

Scene 2: The mapping result of the building basement is shown in Fig. 10, the actual appearance of the scene is shown in Fig. 11, and the statistical data related to the mapping are

(a) (b)

Fig. 7. (Color online) Exterior views of the intelligent wheelchair: (a) side and (b) rear views.

Fig. 8. SLAM result of the second floor of the Engineering Building.

(a) (b)

Fig. 9. (Color online) Images of the second floor of the Engineering Building: (a) second-floor corridor and (b) second-floor hallway.

Table 1
SLAM mapping-related data table for the second floor of the Engineering Building.

Item	Data
Mapping distance (m)	203.243
Total number of words in visual vocabulary	630525
Total number of frames	1050
Short-term memory nodes	10
Working memory nodes	1498
Long-term memory nodes	427
Number of loop closures successes	3

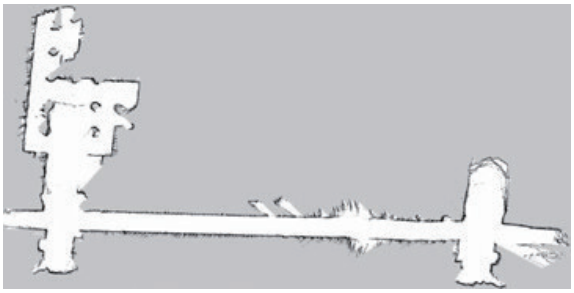


Fig. 10. SLAM result of the basement of the Engineering Building.

(a) (b)

Fig. 11. (Color online) Images of the basement of the Engineering Building: (a) basement staircase and (b) basement hallway.

shown in Table 2. During the process, the intelligent wheelchair returned to the area where the map had been built four times, so it was judged that the loop was successfully closed four times.

The comparison of the mapping data between the second floor and the basement of the Engineering Hall is shown in Table 3. Since the light in the basement is usually dim, more computing resources are required for feature extraction, resulting in a 33.14% increase in the time required. However, the basement environment is relatively simple, without too many obstacles and light changes, so the time spent on creating new visual words and creating each frame is slightly lower than that on the second floor.

Table 2
SLAM mapping-related data table for the basement of the Engineering Building.

Item	Data
Mapping distance (m)	140.414
Total number of words in visual vocabulary	354702
Total number of frames	550
Short-term memory nodes	10
Working memory nodes	870
Long-term memory nodes	132
Number of loop closures successes	4

Table 3
SLAM mapping-related data comparison table between the second floor and basement of the Engineering Building

Item	Second floor	Basement	Growth rate (%)
Total number of words in visual vocabulary	818	867	5.99
Feature extraction time per frame (ms)	17.26	22.98	33.14
Time required to create new visual words per frame (ms)	131.61	122.98	−6.55
Time required to create per frame (ms)	157.91	144.34	−8.59

4.2 Navigation results with path planning algorithm

In the navigation task, we use two types of corridor on the second floor as well as the basement, all located within the Engineering Building, to demonstrate the results of the planned path. In these environments, the intelligent wheelchair will use the D* Lite algorithm for global path planning based on the pre-constructed SLAM map and the DWA algorithm for local path planning in combination with real-time sensor data, ultimately achieving the goal of autonomous navigation.

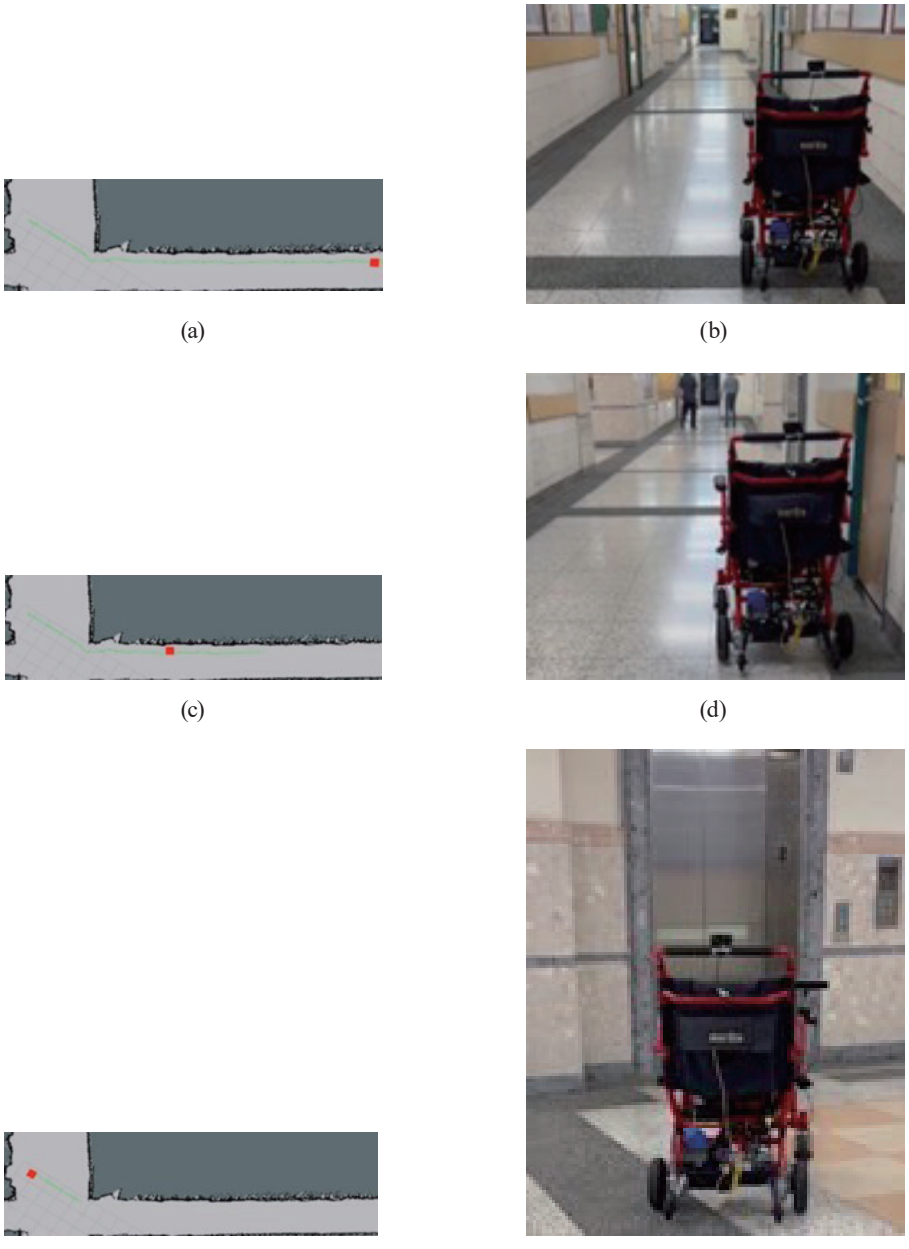
Scene 1: Corridor

In this scenario, the target point was set at the elevator entrance, while the intelligent wheelchair started from the corridor outside the classroom to simulate the task of a person with limited mobility traveling to various facilities. The target point configuration is illustrated in Fig. 12, and while corresponding navigation process is shown in Fig. 13.

Scene 2: Basement

In this experiment, the target point was set in the corridor outside the restroom, allowing the intelligent wheelchair to start from the middle of the corridor and simulate the movement of a person with limited mobility navigating toward the restroom area. The target point configuration is illustrated in Fig. 14, while the corresponding navigation process is depicted in Fig. 15. As shown in Figs. 15(c) and 15(d), a person suddenly appears in front of the intelligent wheelchair. Upon detecting this obstacle, the wheelchair halts and evaluates the appropriate next action.

Fig. 12. Diagram of target point setup in the corridor.



(a) (b)

(c) (d)

(e) (f)

Fig. 13. (Color online) Navigation process diagram from the corridor to the elevator: (a) calculate the path to the target point, (b) navigation start, (c) process of driving to the target point, (d) during the drive to the target point, (e) complete navigation task, and (f) complete navigation task.

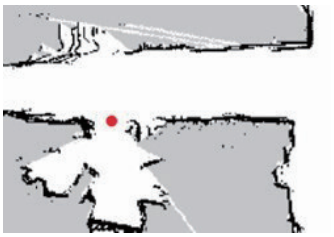
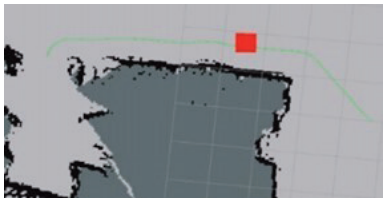
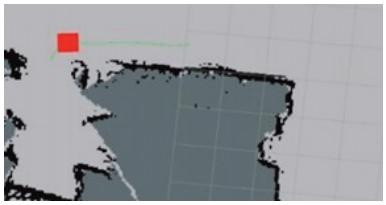


Fig. 14. (Color online) Diagram of target point setup in front of the restroom.

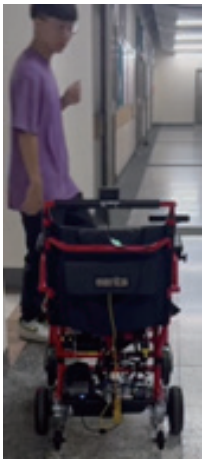


(a)

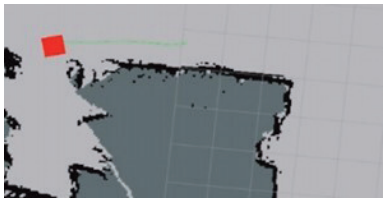
(b)



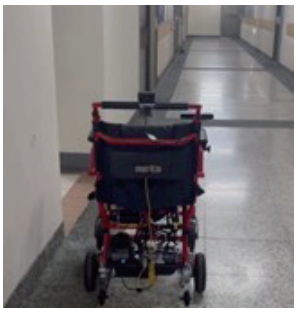
(c)



(d)



(e)



(f)

Fig. 15. (Color online) Navigation process diagram for the basement: (a) calculate the path to the target point, (b) during the drive to the target point, (c) sudden appearance of a person, (d) wheelchair stopping action, (e) complete navigation task, and (f) complete navigation task.

5. Conclusions

In this paper, we presented the development of an autonomous navigation intelligent wheelchair system. Through the hardware modification and sensor integration of a conventional electric wheelchair, a mobile platform with environmental perception and autonomous navigation capabilities was successfully implemented. The research framework is divided into three main components: (1) hardware modification and sensor configuration, (2) map construction based on SLAM, and (3) the design and application of path planning algorithms.

In terms of hardware, the system incorporates sensing modules such as depth cameras and LiDAR sensors, which have been experimentally validated to provide stable and high-precision environmental data. For map construction, the RTAB-Map algorithm is employed to effectively fuse multi-sensor data and utilize the storage management mechanisms of WM and LTM to reduce computational overhead. Regarding navigation, the system integrates the D* Lite algorithm for global path planning with the DWA algorithm for local path planning, enabling dynamic re-planning when encountering moving obstacles, thereby enhancing both flexibility and safety during navigation.

In conclusion, the proposed system markedly improves the navigation stability and reliability of intelligent wheelchairs operating in complex environments and demonstrates strong potential for practical applications in assistive mobility and healthcare settings.

Acknowledgments

This work was supported by the National Science and Technology Council (NSTC) of Taiwan under contract number NSTC 114-2221-E-167-025 (duration: August 1, 2025–July 31, 2026).

References

1 United Nations: https://www.un.org/en/global-issues/population (accessed August 2023).
2 United Nations: https://www.un.org/development/desa/pd/sites/www.un.org.development.desa.pd/files/wpp2022_summary_of_results.pdf (accessed August 2023).
3 Tesla: https://www.tesla.com/fsd (accessed August 2023).
4 M. Milford and A. George: Springer Tracts in Advanced Robotics 92 (Springer, 2014) 38. https://doi.org/10.1007/978-3-642-40686-7_38
5 R. C. Smith and P. Cheeseman: The Int. J. Rob. Res. **5** (1986) 56. https://doi.org/10.1177/027836498600500404
6 H. Longuet-Higgins: Nature **293** (1981) 133. https://doi.org/10.1038/293133a0
7 B. L. E. A. Balasuriya, B. A. H. Chathuranga, B. H. M. D. Jayasundara, N. R. A. C. Napagoda, S. P. Kumarawadu, D. P. Chandima, and A. G. B. P. Jayasekara: Proc. 2016 Moratuwa Engineering Research Conf. (MERCon) (IEEE, 2016) 403–408. https://doi.org/10.1109/MERCon.2016.7480175
8 N. Yu and B. Zhang: Proc. 2018 5th IEEE Int. Conf. Cloud Computing and Intelligence Systems (CCIS) (IEEE, 2018) 279–284. https://doi.org/10.1109/CCIS.2018.8691198
9 R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós: IEEE Trans. Rob. **31** (2015) 1147. https://doi.org/10.1109/TRO.2015.2463671
10 L. Mathieu and M. François: J. Field Rob. **36** (2019) 416. https://doi.org/10.1002/rob.21831
11 L.-C. Chen and L.-H. Lee: Proc. 12th Canadian Symp. Remote Sensing Geoscience and Remote Sensing Symp. (IEEE, 1989) 450–453. https://doi.org/10.1109/IGARSS.1989.578736
12 P. Hart, N. J. Nilsson, and B. Raphael: IEEE Trans. Syst. Sci. Cybern. **4** (1968) 100. https://doi.org/10.1109/TSSC.1968.300136
13 S. Koenig and M. Likhachev: IEEE Trans. Rob. **21** (2005) 354. https://doi.org/10.1109/TRO.2004.838026

14 S. Koenig, M. Likhachev, and D. Furcy: Artif. Intell. **155** (2004) 93. https://doi.org/10.1016/j.artint.2003.12.001
15 D. Fox, W. Burgard, and S. Thrun: IEEE Rob. Autom. Mag. **4** (1997) 23. https://doi.org/10.1109/100.580977
16 Q. Li, R. Li, K. Ji, and W. Dai: Proc. 2015 8th Int. Conf. Intelligent Networks and Intelligent Systems (ICINIS) (IEEE, 2015) 74. https://doi.org/10.1109/ICINIS.2015.35
17 L. Mathieu and M. François: J. Field Rob. **36** (2019) 416. https://doi.org/10.1002/rob.21831
18 H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool: Comput. Vision Image Understanding **110** (2008) 346.
19 F. Zeng, Z. Huang, and Y. Ji: IEEE Signal Process Lett. **24** (2017) 907. https://doi.org/10.1109/LSP.2017.2698140
20 Wikipedia: https://en.wikipedia.org/wiki/K-means_clustering (accessed August 2023).
21 GTSAM: https://gtsam.org/tutorials/intro.html#listing_OdometryOptimize (accessed August 2023).